# Deep Learning para Business Analytics

## Tema A — Del modelo al agente

*Anatomía, modos de trabajo y, sobre todo, cuánto cuesta.*



**Eduardo C. Garrido-Merchán**

Profesor Propio Adjunto, Métodos Cuantitativos, ICADE

ecgarrido@comillas.edu

*Madrid, 2026*

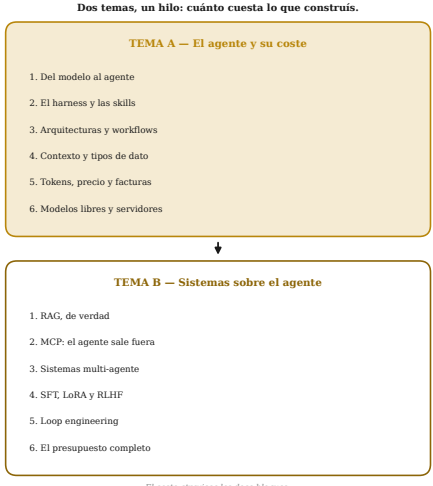
COMILLAS UNIVERSIDAD PONTIFICIA ICAI ICADE CIHS

Hugging Face

NVIDIA

*herramientas del tema*

# Lo que cubre este tema.




**De dónde venimos.** Ya sabéis qué es un Transformer y qué es un LLM. Aquí no se añade arquitectura nueva: se añade todo lo que convierte a ese modelo en algo que trabaja.

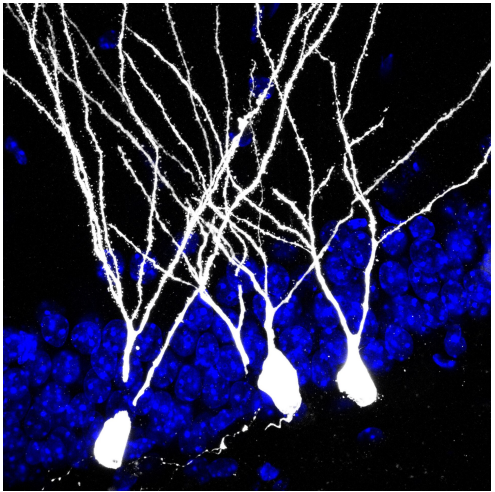
**Al final del tema sabréis.** Montar un agente, decidir entre workflow y agente, elegir modelo con criterio y calcular, con números, lo que costará al mes.

**El hilo conductor.** Cada bloque termina en una línea del presupuesto. El proyecto final se defiende ante inversores, y la primera pregunta de un inversor es el coste por usuario.

**Idea clave.** Este tema se evalúa en el Caso II y en el modelo de costes de la memoria final.

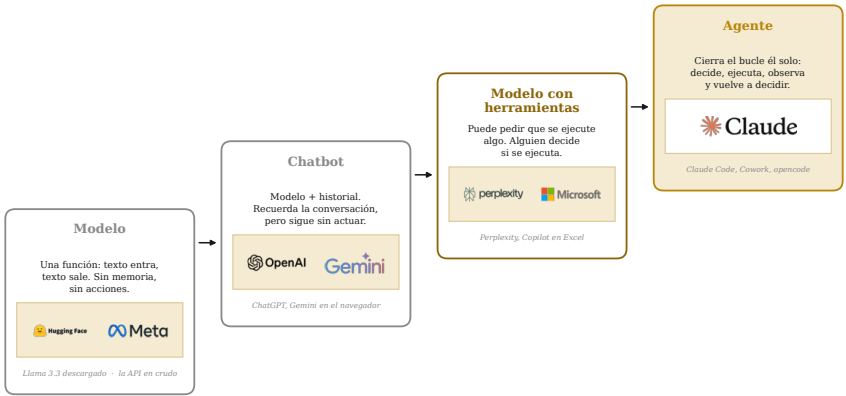
Parte 1 / 6

# Del modelo al agente



# Un modelo es una función; un agente es un bucle alrededor de ella.

**Lo único que cambia de verdad: quién decide el siguiente paso.**




*Los cuatro usan el mismo modelo. Lo que cambia es cuánto harness le han puesto alrededor.*

# Las cuatro piezas que distinguen al agente del chatbot.




**1. Herramientas.** El modelo puede pedir que se ejecute algo fuera de él: leer un fichero, lanzar una consulta SQL, llamar a una API.

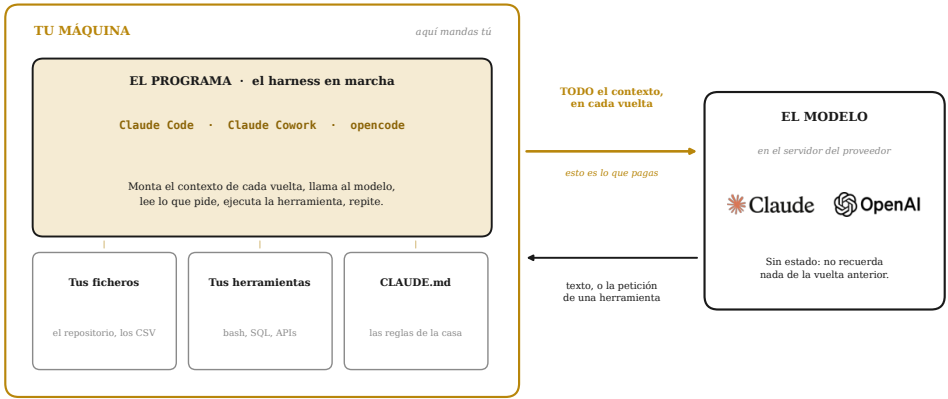
**2. Bucle.** Tras cada acción vuelve a mirar el resultado y decide otra vez. Ese «otra vez» es lo que lo hace agente.

**3. Criterio de parada.** Decide él cuándo ha terminado, o cuándo necesita preguntaros algo.

**4. Permisos.** Alguien ha tenido que decidir de antemano qué puede ejecutar sin pedir confirmación.

**Idea clave.** Quitadle cualquiera de las cuatro y vuelve a ser un chatbot con buena prensa.

# Hay un programa que ejecuta el bucle, y está en vuestra máquina.




**Tus ficheros los abre el programa, y es él quien ejecuta las herramientas. A ese programa lo llamamos harness.**

# Tres puertas al mismo harness ⚙️.




**Claude Code.** En el terminal, dentro de un repositorio. Es el que usaremos en clase.

```
npm install -g @anthropic-ai/claude-code
```

**Claude Cowork.** El mismo motor sin terminal, sobre carpetas en vez de repositorios, pensado para trabajo de oficina: documentos, hojas de cálculo, investigación. Es la puerta que usará la mayoría de vuestros compañeros de empresa.

**opencode.** Libre y agnóstico del proveedor: le enchufáis la API que queráis, o un modelo local.

```
curl -fsSL https://opencode.ai/install | bash
```

**Lo que comparten.** Los tres hacen lo mismo: montar el contexto, llamar al modelo, ejecutar herramientas, repetir. Cambia la puerta de entrada y quién paga la llamada.

**Idea clave.** Cuando alguien os venda «una IA», preguntad cuál de estas tres cosas es.

# Instalar las cuatro piezas: una orden por sistema.

| | Windows 10/11 | macOS 13+ | Ubuntu 22.04+ |
|---|---|---|---|
| **Claude Code**<br>*el agente en la terminal* | **PowerShell**<br>`irm https://claude.ai/install.ps1 \| iex` | **script oficial · macOS, Ubuntu y WSL**<br>`curl -fsSL https://claude.ai/install.sh \| bash` | |
| **Claude**<br>*la app de escritorio: pestañas Chat, Cowork y Code* | **instalador .exe**<br>`claude.com/download` | **imagen .dmg**<br>`claude.com/download` | **repositorio apt (beta)**<br>`sudo apt install claude-desktop` |
| **opencode**<br>*el agente libre, con el proveedor que elijáis* | **npm, o WSL con el script**<br>`npm install -g opencode-ai` | **script oficial · macOS y Ubuntu**<br>`curl -fsSL https://opencode.ai/install \| bash` | |
| **Ollama + Qwen3**<br>*el modelo que corre dentro de vuestro portátil* | **winget, sin Administrador**<br>`winget install Ollama.Ollama` | **imagen .dmg**<br>`ollama.com/download` | **script oficial**<br>`curl -fsSL https://ollama.com/install.sh \| sh` |

Comprobación, en este orden: `claude --version` · `opencode --version` · `ollama --version`

# Nada de esto toca la BIOS: todo aterriza en vuestra carpeta de usuario.




**La pregunta de siempre.** Ninguno de los cuatro instaladores toca el arranque, ni las particiones, ni el firmware. Escriben en vuestra carpeta de usuario y añaden un comando al PATH.

**Administrador.** No hace falta en Windows ni en macOS. En Ubuntu lo pide `apt`, igual que para cualquier otro programa.

**La única excepción, y se esquiva.** Cowork en Ubuntu levanta una máquina virtual y pide virtualización por hardware activada en el firmware. Si os sale ese aviso, no entréis en la BIOS: usad Claude Code en la terminal, que ejecuta el mismo motor. En Windows y macOS el aviso no aparece.

**Espacio.** Unos 4 GB para los programas y entre 2,5 y 5,2 GB más por cada modelo de Ollama que os bajéis.

**Idea clave.** Si algo sale mal, se desinstala borrando una carpeta. No hay nada que «se pueda cargar» el portátil.

# Claude Code: una orden, una comprobación y dentro.

*instalación — Claude Code*

```
# Windows, en PowerShell
PS> irm https://claude.ai/install.ps1 | iex

# macOS, Ubuntu y WSL
$ curl -fsSL https://claude.ai/install.sh | bash

$ claude --version
  2.1.211 (Claude Code)
$ claude doctor

$ cd ~/ventas
$ claude
```

**Requisitos.** macOS 13+, Windows 10 1809+ o Ubuntu 20.04+, y 4 GB de RAM. Nada más.

**Cuenta.** Pide plan Pro, Max, Team, Enterprise o una cuenta de Console con crédito. El plan gratuito de claude.ai no incluye Claude Code.

**Si el comando no aparece.** Cerrad la terminal y abrid otra: el directorio de instalación se añade al PATH y las terminales ya abiertas no lo ven.

**Si os quejáis de `&&` o de `irm`.** Habéis pegado el comando de CMD en PowerShell o al revés. El prompt lo dice: `PS C:\` es PowerShell.

**Alternativas.** `winget install Anthropic.ClaudeCode`, `brew install --cask claude-code`, o `npm install -g @anthropic-ai/claude-code` con Node 22+. Nunca con `sudo`.

# Claude de escritorio: Cowork y Code son dos pestañas de la misma app.

*instalación — Claude (escritorio)*

```
# Windows y macOS
claude.com/download
  .exe en Windows · .dmg en macOS
  instalar, abrir, iniciar sesión

# Ubuntu 22.04+ · beta
  1. clave y repositorio apt de
     Anthropic (bloque en la guía)

$ sudo apt update
$ sudo apt install claude-desktop
$ claude-desktop
```

**Qué instaláis.** Una sola aplicación con tres pestañas: *Chat* para conversar, *Cowork* para encargos largos sobre carpetas y *Code* para repositorios.

**El paso 1 de Ubuntu.** Son dos órdenes largas que no caben en una slide sin partirlas mal. Están, para copiar y pegar, en la guía de instalación de la web, y en la fuente original: `code.claude.com/docs/en/desktop-linux`.

**Cuenta.** Se entra con la suscripción de claude.ai o con el SSO de la universidad. Cowork necesita plan de pago; la pestaña Chat, no.

**Ubuntu y la virtualización.** Si la pestaña Cowork avisa de que necesita KVM, dejadla. No vamos a tocar el firmware por una pestaña: la terminal hace lo mismo.

**Idea clave.** Vuestros compañeros de empresa entrarán por aquí, no por la terminal. Conviene haberlo visto.

# Ollama: el modelo se baja una vez y luego funciona sin red 🔌.

*instalación — Ollama y Qwen3*

```
# macOS y Ubuntu
$ curl -fsSL https://ollama.com/install.sh
| sh

# Windows, sin Administrador
PS> winget install Ollama.Ollama

$ ollama pull qwen3:8b
  5,2 GB · se baja una sola vez
$ ollama run qwen3:8b

$ ollama rm qwen3:8b
  así se recupera el disco
```

**Qué modelo elegir.** Con 16 GB de RAM, `qwen3:8b` (5,2 GB). Con 8 GB, `qwen3:4b` (2,5 GB), que además trae ventana de 256k. Si el portátil va justo, `qwen3:1.7b` (1,4 GB) sigue sirviendo para ver el bucle.

**Dónde cae.** Los pesos van a `~/.ollama` en macOS y Ubuntu, y a `%HOMEPATH%\.ollama` en Windows. Borrar esa carpeta lo deja todo como estaba.

**Sin GPU también funciona.** En CPU va más lento, no deja de funcionar. La GPU acelera; no es requisito.

**Señal de que os habéis pasado.** Si el portátil empieza a tirar de disco y cada respuesta tarda minutos, el modelo no cabe en RAM. Bajad de talla: eso también es dimensionar.

# opencode habla con Ollama en cuanto le dais la dirección.

*~/ventas/opencode.json*

```
{
  "provider": { "ollama": {
    "npm": "@ai-sdk/openai-compatible",
    "name": "Ollama (local)",
    "options": { "baseURL":
        "http://localhost:11434/v1" },
    "models": { "qwen3:8b": {} }
  } }
}
```

**Antes del fichero.** `curl -fsSL https://opencode.ai/install | bash` en macOS y Ubuntu; `npm install -g opencode-ai` en Windows.

**Qué dice el fichero.** Que existe un proveedor llamado `ollama`, que se habla con él con el protocolo de OpenAI y que escucha en el puerto 11434 de vuestra propia máquina. Nada sale del portátil.

**Cómo se elige el modelo.** Dentro de la sesión, `/models`. Desde fuera, `opencode --model ollama/qwen3:8b`.

**El mismo fichero, otro proveedor.** Cambiando esas cinco líneas, opencode habla con Anthropic, con OpenAI o con el servidor de vuestra empresa. El agente no se entera.

**Idea clave.** Con esto tenéis un agente completo, sin cuenta, sin tarjeta y sin conexión. La calidad es menor; la anatomía, idéntica.

# Abrimos un agente sobre un repositorio de verdad.

**Demo en directo · 5 min**

*bash — ~/ventas*

```
$ cd ~/ventas
$ claude

> /status
> ¿qué hay en este directorio?
```

**Lanzo.** Claude Code sobre una carpeta con un CSV de ventas y cuatro scripts. Nada preparado: lo que haya.

**Miráis.** Qué modelo contesta, qué herramientas pide antes de responder, y cuántas vueltas da hasta decidir que ha terminado.

**Si falla.** Si la red falla, lo mismo con `opencode` y el modelo local: más lento, misma anatomía.

**Idea clave.** Lo que acabáis de ver es el harness usando el modelo.

# Qué producto es cada escalón, y qué no puede hacer.

| Escalón | Lo usáis como | Puede | No puede |
|---|---|---|---|
| Modelo | La API en crudo, o Llama 3.3 descargado de Hugging Face. | Transformar el texto que le pasáis. | Recordar nada, ni tocar un fichero. |
| Chatbot | ChatGPT o Gemini en el navegador. | Sostener una conversación larga. | Abrir vuestro ERP ni ejecutar nada. |
| Modelo con herramientas | Perplexity, Copilot dentro de Excel. | Buscar, calcular, rellenar una celda. | Encadenar pasos por su cuenta. |
| Agente | Claude Code, Cowork, opencode. | Decidir cuántos pasos da y ejecutarlos. | Saber qué no debe tocar si no se lo decís. |

**Idea clave.** La diferencia de precio entre la fila 2 y la fila 4 es de dos órdenes de magnitud. La de capacidad, también.

# Una herramienta es un contrato JSON 🔧, no magia.

**1 · Tú declaras la herramienta**

*lo escribes una vez*

```
{
  "name": "consultar_stock",
  "description":
      "Stock por SKU",
  "input_schema": {
    "sku": {"type": "string"}
  }
}
```

**2 · El modelo la pide**

*no ejecuta: pide*

```
{
  "type": "tool_use",
  "name": "consultar_stock",
  "input": {
    "sku": "ZR-4410"
  }
}
```

**3 · Tu código responde**

*aquí ejecutas tú*

```
{
  "type": "tool_result",
  "content":
      "128 uds en Zaragoza,
       0 en Madrid"
}
```

**~ 90 tokens de entrada**

y se pagan en TODAS las llamadas, se use la herramienta o no

**~ 40 tokens de salida**

se facturan al precio caro: los genera el modelo

**~ 30 tokens de entrada**

y vuelven a viajar en cada vuelta posterior del bucle

**El modelo solo pide; ejecuta tu código. Esa frontera es donde se ponen las barandillas.**

# Una llamada a herramienta, en directo y sin cortes.

**Demo en directo · 4 min**



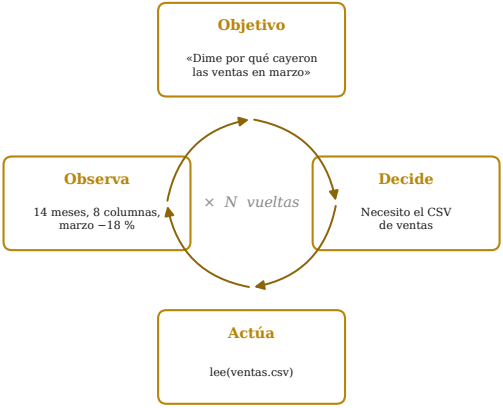

**Lanzo.** La misma pregunta de la figura anterior, para ver el JSON de ida y vuelta tal cual sale.

**Miráis.** Que el modelo *pide* la herramienta y se queda esperando. El fichero lo abre el programa.

**Si falla.** Si no tengo red, enseño el mismo intercambio en el registro de la sesión anterior.

**Idea clave.** Entre lo que el modelo pide y lo que se ejecuta hay una frontera. Ahí es donde se ponen las barandillas.

# El bucle, sobre una pregunta de negocio real.




**Lo que nadie cuenta la primera vez**

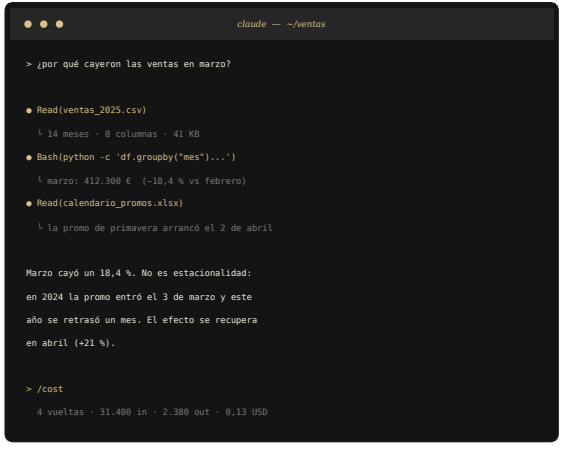
Cada vuelta del bucle reenvía
TODO el contexto anterior.

```
Vuelta 1:    4.000 tokens
Vuelta 2:    6.500 tokens
Vuelta 3:    9.800 tokens
Vuelta 4:  14.200 tokens
           ──────────────
Total:     34.500 tokens
```

No pagas la última llamada.
Pagas la suma de todas.

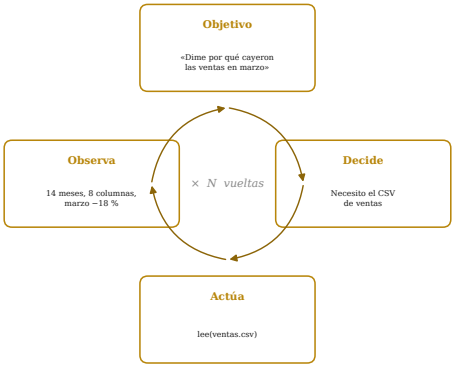
*La caché de prompt decide aquí si el proyecto es viable: descuenta el 90 % de todo lo que se repite entre vueltas.*

# El mismo bucle, tal y como se ve en pantalla.




*Mismo bucle de la figura anterior, esta vez con los tokens contados por el propio programa.*

# Por qué el coste de un agente sorprende siempre la primera vez.




*Cada vuelta reenvía todo lo anterior.*

**La API no tiene memoria.** Cada llamada es independiente. Para que el modelo «recuerde» la vuelta anterior, el harness le reenvía toda la conversación.

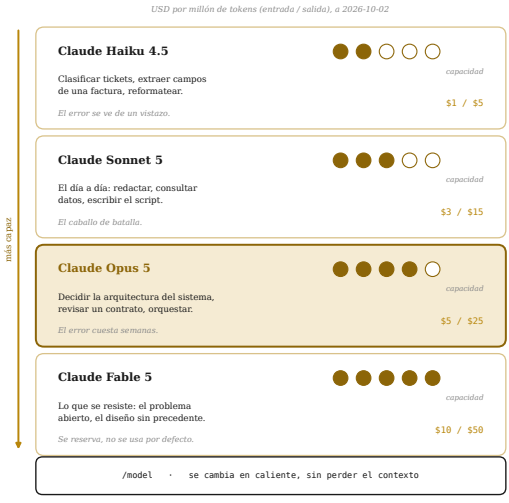
**Consecuencia.** Una sesión de *N* vueltas no cuesta *N* llamadas: cuesta la suma de *N* contextos que crecen.

**Ejemplo a mano: Cuatro vueltas de 4.000 tokens iniciales**

Si cada vuelta añade 2.500 tokens, pagáis 4,000 + 6,500 + 9,000 + 11,500 = 31,000 tokens de entrada, no 4.000.

**Idea clave.** La factura crece de forma cuadrática con la longitud de la sesión si no se gestiona el contexto.

# Elegir el cerebro: capacidad donde equivocarse salga caro.




**La pieza intercambiable.** El mismo harness, el mismo fichero de instrucciones y las mismas herramientas funcionan con los cuatro. Lo que cambia al elegir es cuánto razona y cuánto cuesta cada vuelta del bucle.

**El criterio.** Pagad capacidad donde el error sea caro. Una etiqueta mal puesta en un ticket se corrige en diez segundos; una arquitectura mal decidida se paga durante meses.

**Se decide tarea a tarea.** `/model` cambia de tier a mitad de sesión sin perder el contexto: se sube para el paso difícil y se vuelve a bajar para el resto.

**Idea clave.** El sobrecoste del modelo potente en los tres pasos decisivos es ruido frente a rehacer el trabajo entero.

# Ocho comandos que vamos a usar durante todo el tema.

| Comando | Qué te enseña | Qué concepto del tema mide |
|---|---|---|
| `/status` | Modelo activo, cuenta, directorio. | *El harness, de un vistazo.* |
| `/context` | Cuánto ocupa cada capa de la ventana. | *Las cuatro capas del presupuesto.* |
| `/compact` | Resume el historial y lo sustituye. | *Podar frente a compactar.* |
| `/usage` | Consumo frente al límite del plan. | *Qué llevas gastado hoy.* |
| `/cost` | Tokens y USD de esta sesión. | *Los cuatro números de la fórmula.* |
| `/model` | Cambia de tier sin salir de la sesión. | *Palanca 1: bajar de modelo.* |
| `/permissions` | Qué se ejecuta solo y qué pregunta. | *Las barandillas.* |
| `/skills` | Qué manuales hay y cuáles están cargados. | *Carga progresiva.* |

En opencode los equivalentes son `opencode stats` · `opencode models` · `/compact`

# Medimos la sesión que acabamos de abrir.

**Demo en directo · 6 min**

*claude — ~/ventas*

```
> /context
  system 2,1k · skills 0,9k
  historial 18,4k · libre 178k
> /cost
> /compact
> /context
```

**Lanzo.** `/context` antes de compactar, `/compact`, y `/context` otra vez sobre la misma sesión.

**Miráis.** Cuánto ocupa cada capa, cuál es la única que crece sola, y cuánto historial se pierde al compactar.

**Si falla.** Si la sesión va corta de historial, pego un PDF largo para inflarla y repito.

**Idea clave.** Esta es la medición que tendréis que repetir en el Caso II; de aquí salen las cifras del presupuesto.

# Qué está pasando ya en las empresas de vuestro sector.

*Expansión · Tecnología* Expansión

**La banca española mide ya el gasto en IA por consulta, no por licencia**

Los equipos de datos de las grandes entidades han dejado de presupuestar la IA como software y han empezado a hacerlo como infraestructura: coste por token, por usuario y por mes.

*Cinco Días · Empresas* CincoDías

**El coste por token baja, pero la factura sube: el efecto del agente**

Cada agente reenvía su contexto en cada vuelta del bucle. Los equipos que no miden tokens descubren el problema cuando llega la factura del segundo mes.

**Lo que un inversor te va a preguntar en el pitch**

¿Cuánto te cuesta atender a UN usuario durante un mes?

¿Qué pasa con ese número si creces 10×?

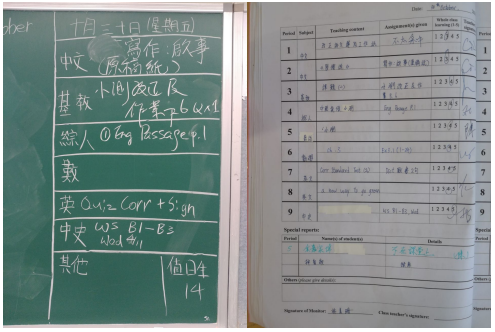
¿Cuál es tu margen si el precio del modelo no baja?

# Concepto → aplicación: dónde encaja un agente en una empresa.

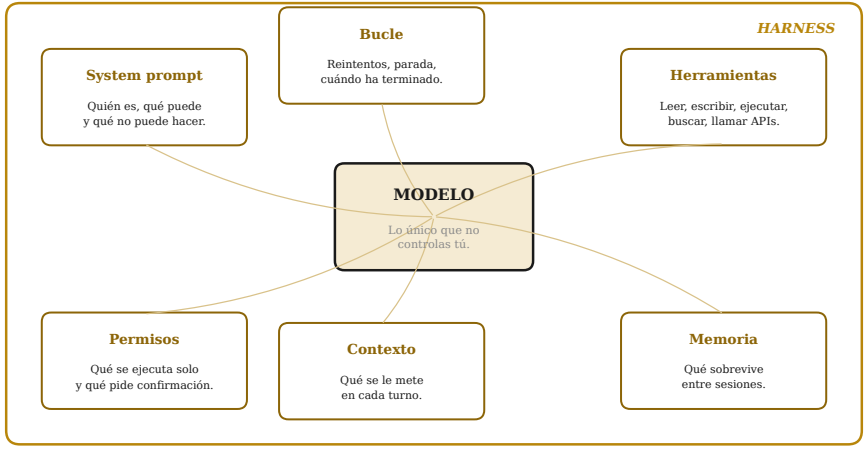
| Función | Caso concreto | Entradas | KPI |
|---|---|---|---|
| Finanzas | Conciliar extractos bancarios con el ERP. | Extractos + asientos. | % conciliado solo. |
| Compras | Comparar ofertas de proveedor y resumir diferencias. | PDFs de oferta. | Horas/expediente. |
| Atención | Responder borradores de tickets con el histórico. | Ticket + base de conocimiento. | % resueltos sin humano. |
| Riesgos | Revisar contratos y marcar cláusulas atípicas. | Contratos en PDF. | Cláusulas detectadas. |
| Datos | Explorar un dataset y proponer el primer informe. | CSV de negocio. | Tiempo hasta el insight. |

**Idea clave.** El patrón común: tareas que mezclan texto, datos y una decisión, repetidas muchas veces al mes.

# El harness y las skills

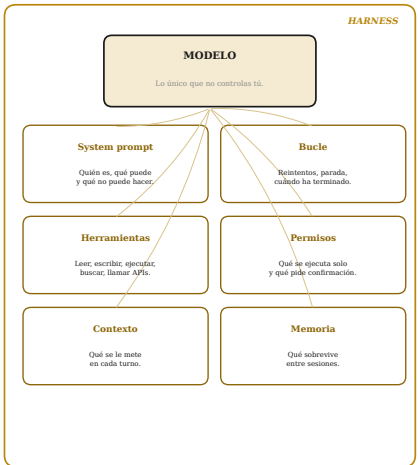


# El harness: todo lo que rodea al modelo ⚙️.




*Cambiar de modelo mueve la calidad un poco. Cambiar el harness lo mueve todo.*

# Las seis piezas del harness, una por una.




*Cambiar de modelo mueve la calidad un poco.*
*Cambiar el harness lo mueve todo.*

**System prompt.** Quién es el agente, qué convenciones sigue, qué tiene prohibido. Es la pieza con mejor relación coste/efecto de todas.

**Bucle.** Cuántas vueltas como máximo, qué hacer si una herramienta falla, cuándo parar.

**Herramientas.** El catálogo de acciones. Cada una con su esquema y su descripción.

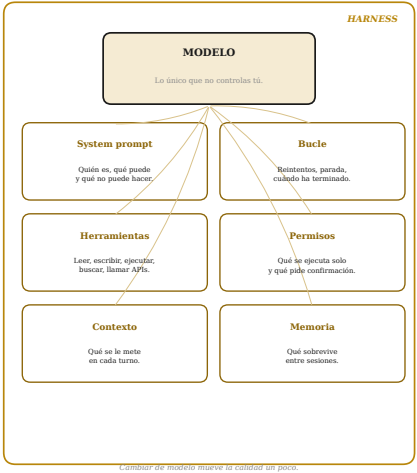
**Permisos.** Qué se ejecuta sin preguntar y qué exige confirmación humana. Aquí vive el riesgo operativo.

**Contexto.** Qué se le mete en cada turno y qué se poda. El capítulo más caro de la factura.

**Memoria.** Qué sobrevive al cierre de la sesión: un fichero, una base de datos, nada.

**Idea clave.** Dos equipos con el mismo modelo y resultados opuestos: la diferencia está casi siempre en el harness.

# El fichero de instrucciones: cinco bloques que valen más que cualquier prompt ingenioso.




*Cambiar de modelo mueve la calidad un poco. Cambiar el harness lo mueve todo.*

**1. Rol.** «Eres el asistente de análisis del equipo de riesgos de una aseguradora española.»

**2. Convenciones.** «Las cifras en euros con separador de miles. Los informes en castellano, registro neutro.»

**3. Restricciones.** «No escribas en `produccion/`. No instales dependencias sin preguntar.»

**4. Contexto del proyecto.** «Los datos están en `data/polizas.parquet`; el diccionario de campos, en `docs/`.»

**5. Política de IA.** «Declara en cada entregable qué parte se generó con IA y conserva los prompts.»

**Idea clave.** Se versiona con el repositorio. Si vive en la cabeza de una persona, no existe.

# Abrimos el harness por dentro y le cambiamos las reglas.

**Demo en directo · 7 min**

*claude — ~/ventas*

```
> /init
  escribe CLAUDE.md
> /skills
> /permissions
  edito CLAUDE.md y repito
> ¿qué convenciones sigues?
```

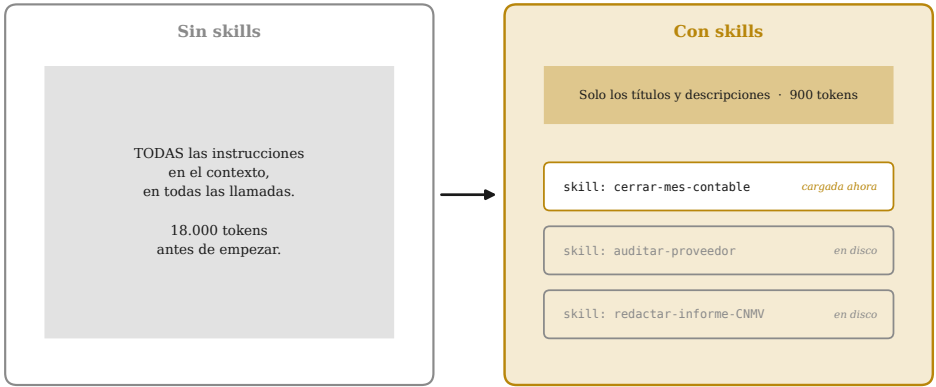
**Lanzo.** `/init` para que escriba él su propio fichero de instrucciones, y luego lo edito delante de vosotros.

**Miráis.** Que la misma pregunta da otra respuesta después de cambiar tres líneas de CLAUDE.md, con el mismo modelo.

**Si falla.** Si `/init` tarda, uso el CLAUDE.md que ya tengo escrito y enseño el diff.

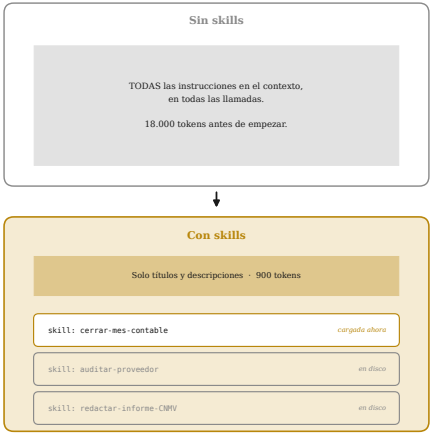
**Idea clave.** Esto es lo barato de ajustar. Cambiar de modelo cuesta dinero; cambiar el system prompt cuesta cinco minutos.

# Skills: manuales que el agente abre solo cuando hacen falta.




*El contexto es un presupuesto. Las skills te dejan pagar solo por el manual que estás usando.*

# Por qué las skills son una decisión económica y no sólo de orden.




*El contexto es un presupuesto: pagas solo por el manual que estás usando.*

**El problema.** Una empresa mediana acumula decenas de procedimientos. Metidos todos en el system prompt, se pagan en cada llamada aunque no se usen.

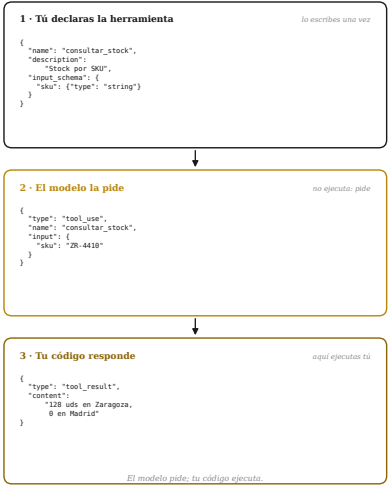
**La carga progresiva.** El agente ve una línea por skill (nombre y para qué sirve). Cuando la tarea encaja, abre el manual completo.

**Ejemplo a mano: 18.000 tokens frente a 900**

Con 20 skills de 900 tokens, cargarlas todas cuesta 18.000 tokens por llamada. Con carga progresiva se pagan 900 de índice y sólo el manual que toque.

**Idea clave.** A 50.000 llamadas al mes y Sonnet 5, esos 17.000 tokens de diferencia son unos 2.550 USD mensuales.

# Las barandillas: dónde se pone el humano en el bucle.




**Regla de reversibilidad.** Lo que se deshace con un `git revert` puede ir solo. Lo que sale de la empresa (un correo, un pago, un borrado) pide confirmación.

**Tres niveles en la práctica.** *Permitir siempre* para leer y buscar; *preguntar* para escribir y ejecutar; *denegar* para lo que nunca debe tocar.

**Dónde se tocan.** `/permissions` lista las reglas vigentes y deja añadirlas sin salir de la sesión. Se versionan en `.claude/settings.json` junto al repositorio.

**Auditoría.** El agente deja traza de cada herramienta que invocó y con qué argumentos. Cuando algo sale mal, se reproduce.

**Idea clave.** El permiso se define por la consecuencia de la acción, no por lo lista que parezca la herramienta.

Parte 3 / 6

# Arquitecturas y workflows



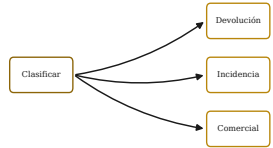
# Cuatro arquitecturas, de la más predecible a la más abierta.

### (a) Cadena fija — un workflow



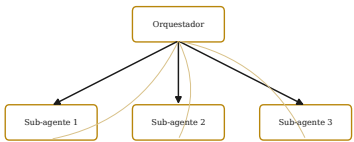

Tú decides el orden. El modelo solo rellena cada paso.
Coste predecible, resultado auditable.

### (b) Router — elige la rama




Un modelo barato enruta; uno caro solo atiende
lo que de verdad lo necesita.

### (c) Orquestador y trabajadores



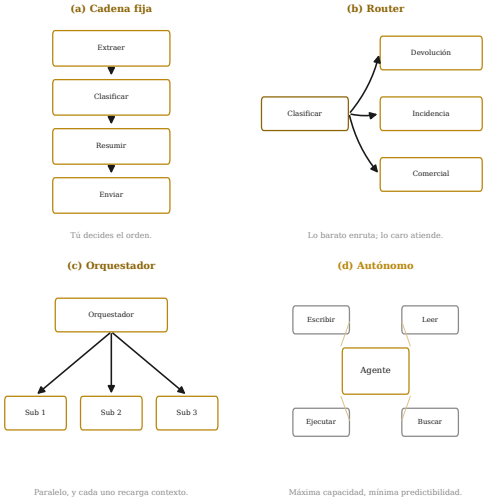

Paralelizas el trabajo. Multiplicas el gasto por el
número de trabajadores: cada uno recarga su contexto.

### (d) Agente autónomo




Máxima capacidad, mínima predictibilidad.
El coste solo se acota poniéndole un límite duro.

# Cómo elegir entre las cuatro.




**Cadena fija.** Cuando conocéis los pasos. Coste y latencia predecibles al céntimo. Es la opción correcta mucho más a menudo de lo que parece.

**Router.** Cuando hay varias ramas conocidas. Un modelo barato clasifica y sólo lo difícil llega al modelo caro: ahorro grande con cambio pequeño.

**Orquestador.** Cuando la tarea se parte en trozos independientes. Ganáis tiempo de reloj y pagáis el contexto de cada trabajador.

**Autónomo.** Cuando ni vosotros sabéis los pasos. Máxima capacidad y el único control de coste es un límite duro de tokens.

**Idea clave.** Subid un escalón sólo cuando el anterior os haya fallado por una razón concreta.

# La mayoría de lo que se vende como «agente» es un workflow.

## WORKFLOW

- Facturas → extraer campos → validar → ERP
- Ticket → clasificar → enrutar → plantilla
- Informe mensual con la misma estructura
- Traducir un catálogo de 8.000 referencias

**Coste predecible.**
**Fallos reproducibles.**
**Barato.**

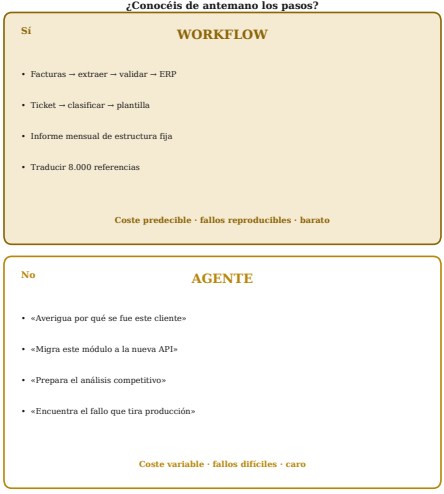
## AGENTE

- «Averigua por qué este cliente se ha ido»
- «Migra este módulo a la nueva API»
- «Prepara el análisis competitivo del sector»
- «Encuentra el fallo que tira producción»

**Coste variable.**
**Fallos difíciles de reproducir.**
**Caro.**

**Sí** — **¿Conoces de antemano los pasos que hay que dar?** — **No**

# El criterio de decisión, en una pregunta.




**La pregunta.** ¿Podéis escribir hoy la lista de pasos que hay que dar? Si podéis, escribidla: sale más barata, más rápida y más fácil de auditar.

**Señales de que necesitáis un agente.** El número de pasos depende de lo que se encuentre por el camino; hay que reaccionar a resultados intermedios; la tarea se describe por objetivo y no por procedimiento.

**Señal de alarma.** Si vuestro «agente» siempre llama a las mismas tres herramientas en el mismo orden, habéis pagado de más por un workflow.

**Idea clave.** El coste de un workflow se calcula. El de un agente se acota.

# Cuatro modos de usar el mismo agente.

### Interactivo

```
claude
```

Conversas, ves cada paso, apruebas lo que toca.

**Para:**

Explorar, depurar, aprender el repo.

### Plan

```
claude  →  shift+tab
```

Te cuenta qué va a hacer antes de tocar nada.

**Para:**

Cambios que dan miedo.

### No interactivo

```
claude -p "..."
```

Un comando, una respuesta, sin conversación.

**Para:**

Scripts, CI, hooks de git.

### Programado

```
cron + claude -p
```

Se ejecuta solo, a su hora, sin nadie mirando.

**Para:**

Informes nocturnos, vigilancia de datos.

# Los modos, con la herramienta en la mano.




**Cloud.** `npm install -g @anthropic-ai/claude-code`, después `claude` dentro del repositorio. Requiere cuenta y crédito.

**Libre.** `curl -fsSL https://opencode.ai/install | bash` y `opencode --model ollama/qwen3:8b`. Cero coste por sesión, calidad menor.

**Interactivo.** Para explorar y para aprender el repositorio. Veis cada paso y aprobáis lo que toca.

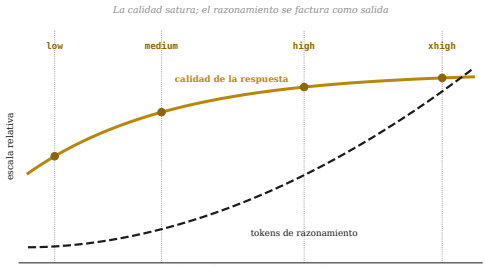
**Plan.** El agente cuenta qué va a hacer antes de tocar nada. Para cambios que dan miedo.

**No interactivo.** `claude -p 'revisa este diff'` dentro de un script o de un *hook* de git. Sin conversación.

**Programado.** El mismo comando bajo `cron`. Informes nocturnos, vigilancia de datos, resúmenes de lunes.

**Idea clave.** El modo programado es el que convierte un juguete en infraestructura, y también el que dispara la factura sin que nadie mire.

# Cuánto razona el modelo antes de contestar también lo decidís vosotros.




**Qué es.** Antes de emitir la respuesta, el modelo puede razonar internamente durante más o menos tiempo. Ese presupuesto se fija en cuatro niveles: `low`, `medium`, `high`, `xhigh`.

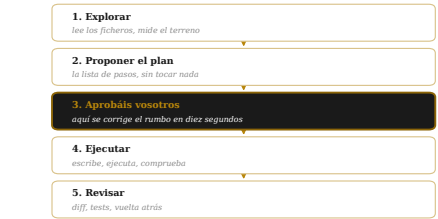
**Por qué aparece en un tema de costes.** Ese razonamiento son tokens de salida, los que se facturan cinco veces más caros que la entrada. Es la partida que no se ve en la respuesta y sí en la factura.

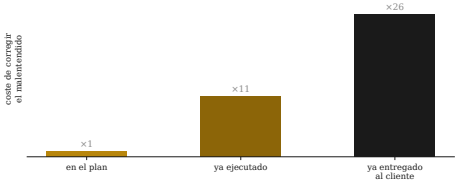
**Cómo se comporta.** La calidad sube con rendimientos decrecientes; el coste sube más deprisa que el esfuerzo. El punto bueno depende de la tarea.

**El modo automático.** Una misma tarea mezcla pasos triviales y pasos decisivos. El reparto automático estima la dificultad de cada paso y le asigna el razonamiento que merece: en la figura, sólo el paso 3 se lo lleva entero.

**Idea clave.** Subidlo cuando el fallo sea caro y difícil de detectar. Bajadlo cuando el fallo salte a la vista.

# Planificar primero: el rumbo se corrige barato mientras el plan sea texto.




*Órdenes de magnitud ilustrativos: lo que importa es que la corrección es casi gratis mientras el plan siga siendo texto.*

**El patrón.** Explorar, proponer un plan, aprobarlo vosotros, ejecutar y revisar. El tercer paso es el único que no se puede automatizar sin perder el control del resultado.

**Cómo se activa.** `shift+tab` dentro de la sesión. Mientras no aprobéis, el agente sólo lee: no escribe ficheros ni ejecuta nada.

**Señal de diagnóstico.** Cuando el plan que propone se parece poco al que teníais en la cabeza, el fallo está casi siempre en cómo formulasteis la tarea. Reformulad y volved a pedirlo: todavía no ha costado nada.

**Idea clave.** Es la barandilla más barata de todas: leer seis líneas antes de que el agente toque el repositorio.

# Los cuatro modos, uno detrás de otro, en dos minutos cada uno.

**Demo en directo · 8 min**

*bash — ~/ventas*

```
$ claude
shift+tab → modo plan

$ claude -p 'resume los
cambios de hoy' > nota.md

$ crontab -e
0 7 * * 1 claude -p ...
```

**Lanzo.** El mismo agente en interactivo, en plan, en `-p` dentro de un *pipe* de shell, y la línea de `cron` que lo programa.

**Miráis.** Que el modo `-p` escupe texto plano y encadena con cualquier herramienta de Unix. Ahí es donde esto deja de ser un chat.

**Si falla.** Si `cron` no está disponible, enseño el fichero de *hook* de git que hace lo mismo al hacer `commit`.

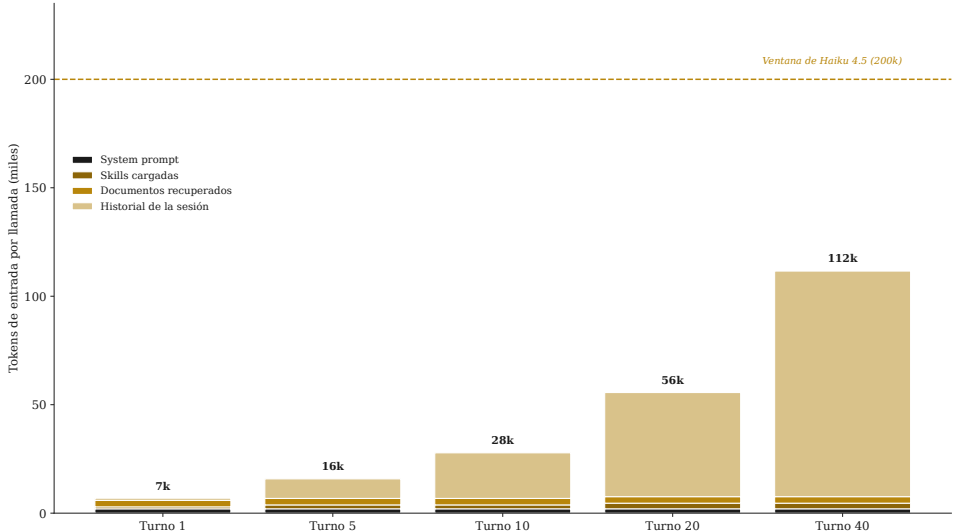
**Idea clave.** El salto de productividad está en el tercer modo. El salto de factura, en el cuarto.

Parte 4 / 6

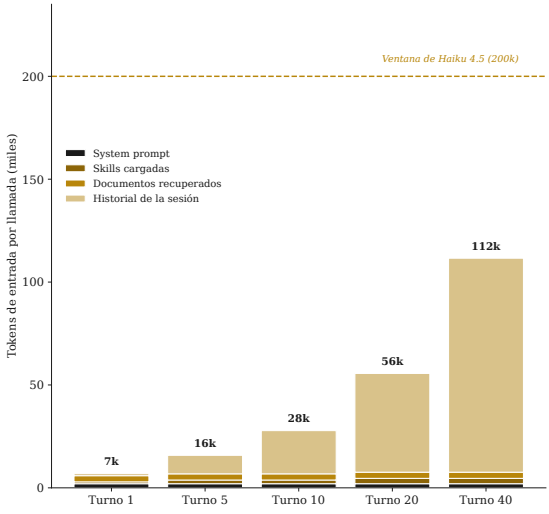
# El contexto y los datos



# El historial se come el presupuesto, y lo pagáis en cada turno.



# Las cuatro capas que compiten por la ventana.




**Restricción.** Las cuatro capas comparten un único presupuesto de tokens:

$$\underbrace{t_{\text{sys}}}_{\text{fijo}} + \underbrace{t_{\text{skills}}}_{\text{bajo demanda}} + \underbrace{t_{\text{rag}}}_{\text{por consulta}} + \underbrace{t_{\text{hist}}}_{\text{crece}} \leq B$$

**Cuál crece.** Sólo la última. Las otras tres las controláis vosotros; el historial crece solo mientras el agente trabaja.

**Las dos salidas.** *Podar* (borrar resultados de herramientas antiguos) o *compactar* (resumir lo anterior). Las dos pierden información; elegid cuál os duele menos.

**Cómo se mide.** `/context` desglosa las cuatro capas de la sesión abierta, y `/compact` ejecuta la segunda salida.

**Idea clave.** Gestionar el contexto es la primera decisión de diseño y la primera partida del presupuesto.

# Qué le podéis dar de comer, y lo que ocupa cada cosa.

| | | |
|---|---|---|
| **Texto plano** | Un folio A4 denso | ~ 700 tokens |
| **PDF** | Memoria anual de 80 páginas | ~ 60.000 tokens |
| **Imagen** | Una captura de pantalla | hasta ~ 4.800 tokens |
| **CSV / Excel** | 20.000 filas de ventas | no cabe: va por herramienta |
| **Audio** | Reunión de una hora | se transcribe primero |
| **Base de datos** | El ERP entero | nunca entra: se consulta |

*Fondo dorado: entra directo en el contexto.*

*Fondo blanco: el agente lo toca con una herramienta, no se lo traga entero.*

**La pregunta de diseño es «¿cuánto me cuesta que quepa, en cada uno de los turnos de la sesión?».**

# La regla para decidir qué entra en el contexto y qué no.




**Entra en el contexto** lo que el modelo necesita *leer entero* para razonar: un contrato, un informe, una captura de pantalla.

**Se toca con una herramienta** lo que el modelo necesita *consultar*: una tabla de 20.000 filas, el ERP, un índice de documentos.

**Ejemplo a mano: El error de principiante**

Pegar un CSV de 50.000 filas en el prompt. Son unos 2 millones de tokens: no cabe, y si cupiera costaría 6 USD por pregunta con Sonnet 5.

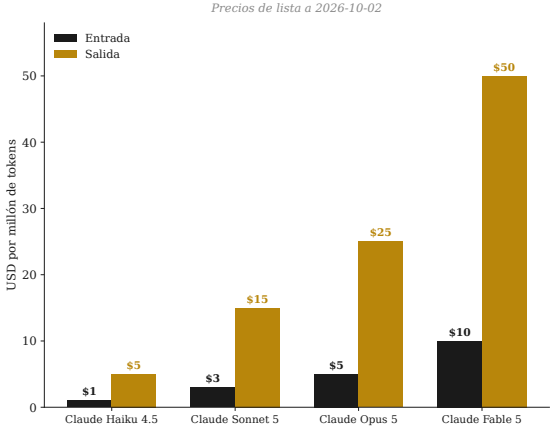
**Idea clave.** Si el dato se puede filtrar antes de llegar al modelo, filtradlo antes: cada token que no mandáis es un token que no pagáis.

Parte 5 / 6

# Tokens, precio y factura



# Qué es exactamente un token, y por qué os lo cobran dos veces.




*Entre el modelo más barato y el más caro hay un factor 10. Elegir modelo es, antes que nada, una decisión de presupuesto.*

**Qué es.** Un trozo de texto de entre uno y varios caracteres. En castellano, unas 0,75 palabras por token de media.

**Dos contadores.** Los tokens de **entrada** (todo lo que le mandáis) y los de **salida** (lo que genera). Se facturan a precios distintos.

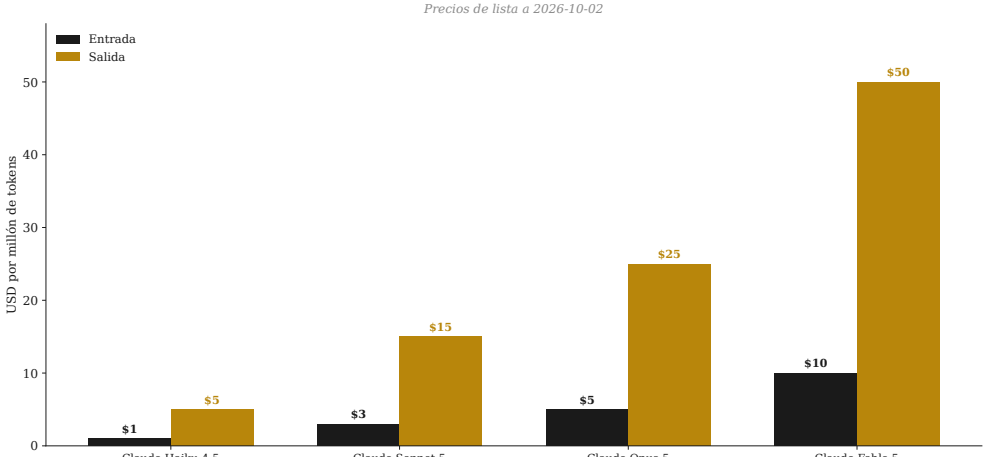
**Por qué la salida es más cara.** Generar exige una pasada del modelo por token. Leer la entrada se paraleliza; escribir la salida es secuencial.

**Ejemplo a mano: De texto a factura**

Un informe de 3 folios son unos 2.000 tokens de salida. Con Opus 5, a 25 USD el millón, son 0,05 USD por informe.

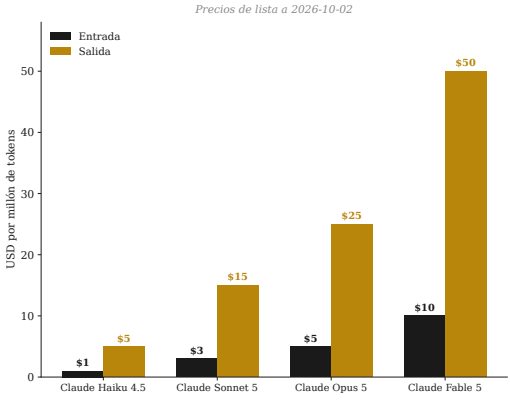
**Idea clave.** Todo presupuesto de IA empieza por estimar estos dos números para una consulta típica.

# Generar cuesta cinco veces más que leer.




*Entre el modelo más barato y el más caro hay un factor 10. Elegir modelo es, antes que nada, una decisión de presupuesto.*

# La fórmula del coste, término a término.




*Entre el modelo más barato y el más caro hay un factor 10. Elegir modelo es, antes que nada, una decisión de presupuesto.*

$$C = p_{\text{in}}\, t_{\text{in}} + p_{\text{out}}\, t_{\text{out}}$$

$$C_{\text{mes}} = C \cdot N$$

$p_{\text{in}}$ precio por token de entrada (USD/MTok $\div 10^6$)

$p_{\text{out}}$ precio por token de salida

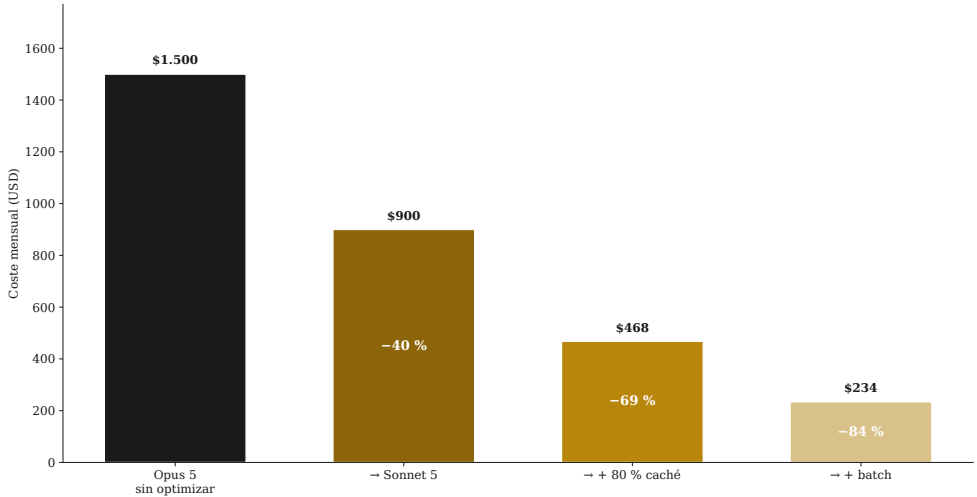
$t_{\text{in}}$, $t_{\text{out}}$ tokens de una consulta típica

$N$ consultas al mes

**Los cuatro números.** Son los únicos que necesitáis para el presupuesto base. Los tres primeros los sabéis; el cuarto es vuestra previsión de negocio.

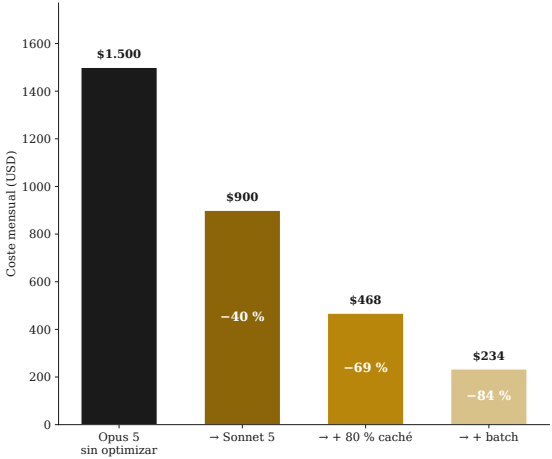
**Idea clave.** Si no sabéis estimar $N$, no tenéis un problema de IA: tenéis un problema de modelo de negocio.

# Tres decisiones, la misma funcionalidad, el 84 % menos de factura.




*50.000 consultas/mes de 4.000 tokens de entrada y 400 de salida. Precios a 2026-10-02.*

# Las tres palancas, explicadas.




*50.000 consultas/mes de 4.000 tokens de entrada y 400 de salida. Precios a 2026-10-02.*

**1. Bajar de tier.** Entre el modelo más caro y el más barato hay un factor 10. Casi ninguna tarea de clasificación necesita el tier alto.

**2. Caché de prompt.** Lo que se repite entre llamadas (system prompt, instrucciones, documentos fijos) se factura al 10 %. Escribirlo en caché cuesta un 25 % más una sola vez.

**3. Batch.** Si la respuesta no tiene que llegar ya, la API de lotes descuenta el 50 %. Entrega en menos de una hora casi siempre, 24 como máximo.

**Idea clave.** Las tres son compatibles entre sí y ninguna cambia lo que el usuario ve.

# Bajamos de tier en caliente y medimos lo que cambia.

**Demo en directo · 6 min**

```
claude — ~/ventas
> /cost
  Opus 5 · 0,34 USD
> /model sonnet
> (repito la misma pregunta)
> /cost
  Sonnet 5 · 0,07 USD
```

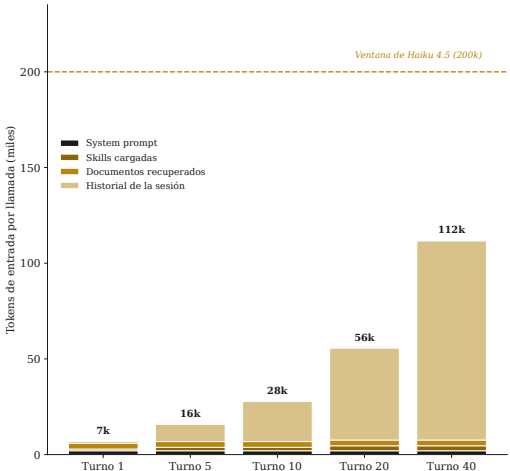
**Lanzo.** La misma pregunta de negocio con el tier alto y con el medio, midiendo con `/cost` las dos veces.

**Miráis.** El factor de precio entre los dos, y si la respuesta ha empeorado lo suficiente para justificarlo. Casi nunca lo hace.

**Si falla.** Si solo tengo un modelo disponible, comparo contra las cifras de la calculadora de la práctica.

**Idea clave.** Esta comparación de dos minutos es la que deberíais hacer antes de fijar el modelo de cualquier proyecto.

# Cómo se rompe la caché sin darse cuenta.



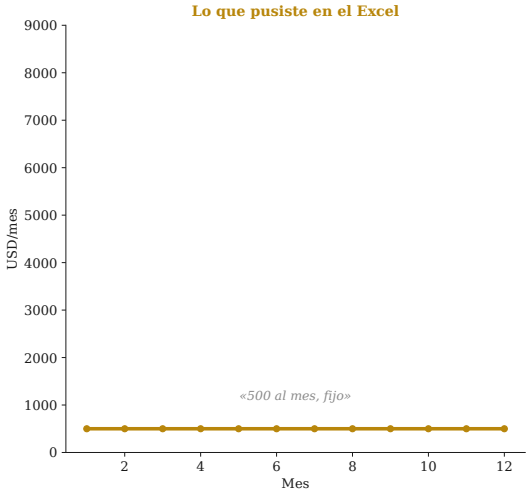

**La regla.** La caché casa por *prefijo exacto*. Un solo byte distinto al principio invalida todo lo que viene detrás.

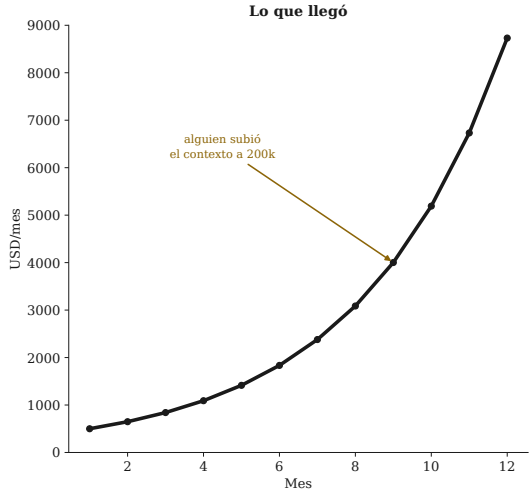
**Los tres asesinos silenciosos.** Meter la fecha y hora en el system prompt; generar un identificador aleatorio por petición; serializar un diccionario sin ordenar las claves.

**Cómo se comprueba.** La respuesta trae un contador de tokens leídos de caché. Si sale cero llamada tras llamada con el mismo prefijo, hay un asesino suelto.

**Idea clave.** Lo estable va delante, lo que cambia va detrás. Ese es todo el diseño.

# El presupuesto de IA: expectativa y realidad.



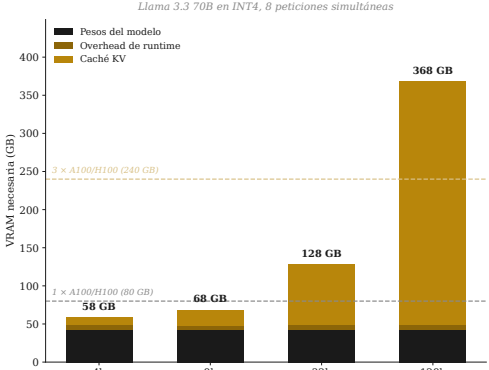



Parte 6 / 6

# Modelos libres y servidores propios



# Qué significa «modelo libre» y cómo se baja uno.




**Libre, con matices.** Los pesos se descargan y se ejecutan donde queráis. La licencia suele permitir uso comercial con condiciones: leedla antes de construir encima.

**Dónde están.** Hugging Face es el repositorio de referencia. Familias habituales: Llama (Meta), Qwen (Alibaba), Mistral, DeepSeek, Gemma (Google).

**Cómo se prueba en cinco minutos.**

```
curl -fsSL https://ollama.com/install.sh | sh
ollama pull qwen3:8b
ollama run qwen3:8b
```

**Cuantización.** Reducir la precisión de los pesos de 16 bits a 4. El modelo ocupa un tercio y pierde poca calidad. `Q4_K_M` es el punto dulce habitual.

**Idea clave.** Probarlo en vuestro portátil es gratis. Servirlo a 500 usuarios no lo es, y ahí empieza el cálculo.

# La misma pregunta, pagando y sin pagar.

**API de pago**

*claude — Sonnet 5 por API*

```
> resume este contrato y marca las
  cláusulas atípicas

● Read(contrato.pdf)  └ 34 págs
● Grep("penalización")  └ 6 hits

Tres cláusulas se salen del estándar:
la 7.2 (penalización al 12 %), la 9.1
(renovación tácita) y la 14 (fuero).

> /cost  └ 0,21 USD · 11 s
```

**Modelo libre en tu portátil**

*opencode — qwen2.5-coder:7b local*

```
> resume este contrato y marca las
  cláusulas atípicas

● read(contrato.pdf)  └ 34 págs

El contrato regula un suministro.
Incluye cláusulas de penalización y
de renovación. Conviene revisarlas.

  (no citó ninguna cláusula concreta)
$ opencode stats  └ 0,00 USD · 94 s
```

**Cero euros y ocho veces más lento, pero la diferencia que importa es que el segundo no citó ni una cláusula.**

# Un agente entero sin tarjeta de crédito.

**Demo en directo · 7 min**

*bash — ~/ventas*

```
$ ollama run qwen3:8b

$ opencode
modelo: ollama/qwen3:8b

> (la pregunta de la demo 1)

$ opencode stats
```

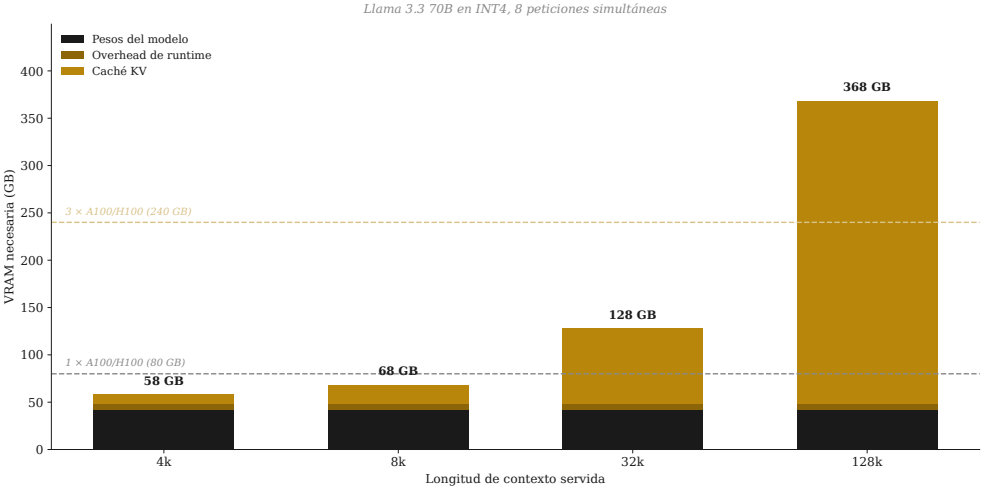
**Lanzo.** La pregunta de la primera demo, esta vez contra un modelo de 8B que corre en este portátil, sin red.

**Miráis.** Que el bucle es idéntico y la factura es cero. Y que a la tercera vuelta se nota de dónde viene la diferencia de precio.

**Si falla.** Si el portátil se queda sin RAM, bajo a `qwen3:4b` y lo digo en voz alta: eso también es dimensionar.

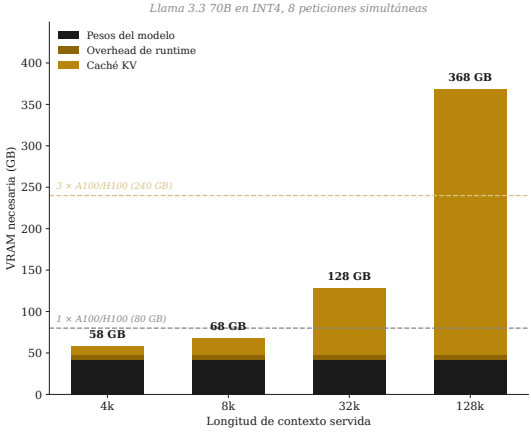
**Idea clave.** Probarlo es gratis. Servirlo a quinientos usuarios es la cuenta de las dos slides siguientes.

# Llama 3.3 70B en INT4: los pesos son lo de menos.




*El modelo cuantizado ocupa 42 GB y no se mueve. La caché KV crece con el contexto y con la concurrencia, y es la que te obliga a comprar más tarjetas.*

# Dimensionar un servidor: la fórmula y la trampa.



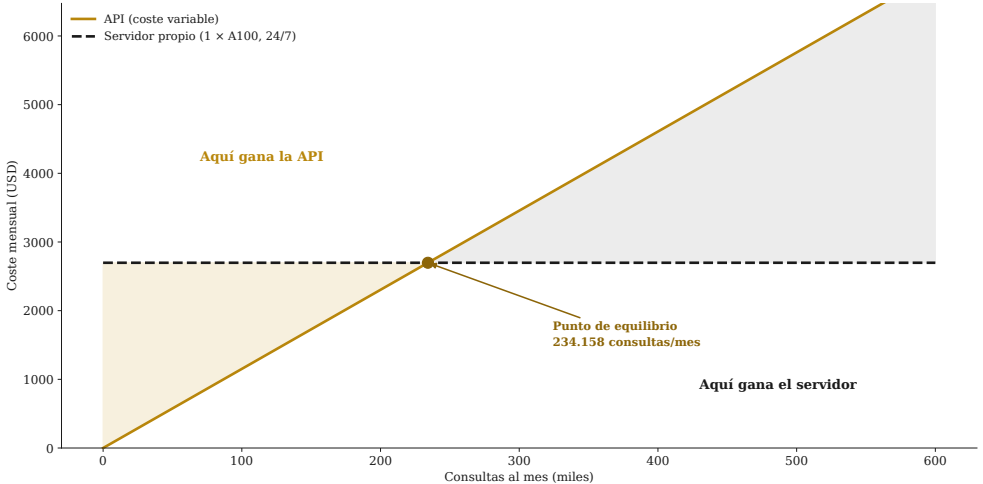

$$\text{VRAM} \approx \underbrace{P \cdot b}_{\text{pesos}} \cdot (1 + \rho) \; + \; \underbrace{k \cdot L \cdot c}_{\text{caché KV}}$$

- $P$ parámetros (en miles de millones)
- $b$ bytes por parámetro: 2 en FP16, 0,6 en INT4
- $\rho$ overhead del runtime, ≈ 0,15
- $k$ MB de caché por cada 1.000 tokens (≈ 320)
- $L$ contexto servido, en miles de tokens
- $c$ peticiones simultáneas

**La trampa.** Casi todo el mundo calcula sólo el primer término. El segundo crece con el contexto $y$ con la concurrencia, y es el que os obliga a comprar la tercera tarjeta.

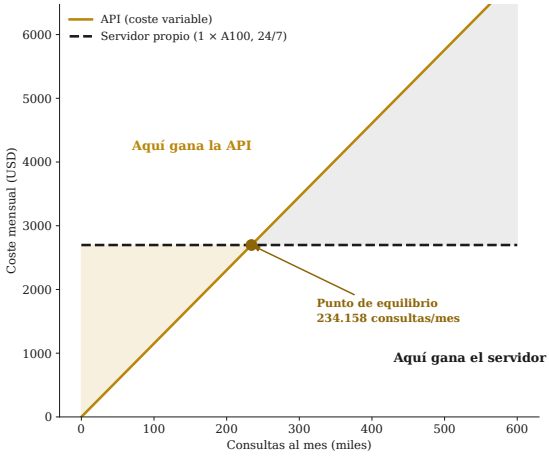
**Idea clave.** Dimensionad con el contexto y la concurrencia reales, no con los del día de la demo.

# El volumen decide si compensa la API o el servidor propio.




*Sonnet 5 con 60 % de caché frente a Llama 3.3 70B en INT4. Incluye un cuarto de jornada de MLOps. Precios a 2026-10-02.*

# Cómo se lee el punto de equilibrio.




*Sonnet 5 con 60 % de caché frente a Llama 3.3 70B en INT4. Incluye un cuarto de jornada de MLOps. Precios a 2026-10-02.*

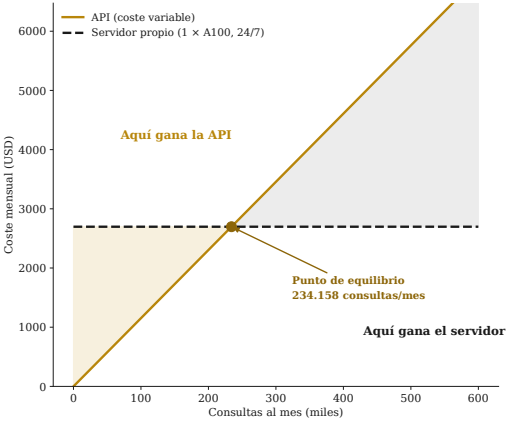
**Dos curvas distintas.** La API es coste *variable* puro: una recta que sale del origen. El servidor es coste *fijo*: una horizontal. Siempre se cortan.

**El umbral.** $N^{\star} = C_{\text{servidor}} / c_{\text{consulta}}$. En este caso, unas 234.000 consultas al mes.

**Lo que inclina la balanza.** Apagar el servidor fuera de horario baja la horizontal un 65 %. Activar la caché baja la pendiente de la recta. Las dos mueven el cruce.

**Idea clave.** El servidor propio se justifica por volumen sostenido, por residencia del dato o por pesos ajustados; casi nunca por precio a volúmenes de arranque.

# Lo que el gráfico no dice: las cuatro partidas que se olvidan.




*Sonnet 5 con 60 % de caché frente a Llama 3.3 70B en INT4. Incluye un cuarto de jornada de MLOps. Precios a 2026-10-02.*

**1. Personal.** Un servidor en producción necesita alguien que lo vigile. Un cuarto de jornada de MLOps son unos 1.240 USD al mes, y ya está dentro de la horizontal del gráfico.

**2. Redundancia.** Una sola tarjeta significa una sola avería hasta la caída del servicio. El doble de tarjetas, el doble de horizontal.

**3. Actualización.** Los modelos libres mejoran cada pocos meses. Migrar, reevaluar y volver a desplegar es trabajo recurrente.

**4. Ocupación.** Un servidor cuesta lo mismo al 5 % que al 95 % de uso. La API cuesta cero cuando nadie pregunta.

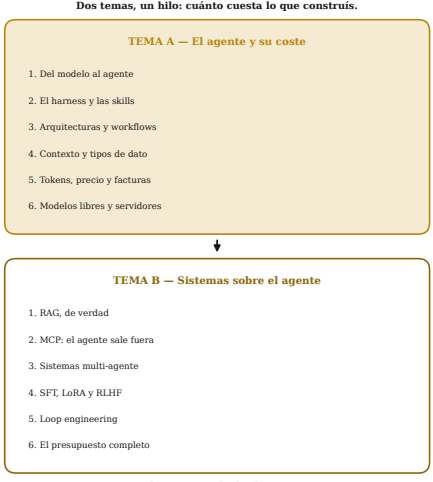
**Idea clave.** Comparad coste total de propiedad contra coste total de propiedad, no precio por token contra precio por hora.

# Cuándo elegir cada vía, en una tabla.

| Criterio | API de pago | Modelo libre en servidor |
|---|---|---|
| Volumen bajo o irregular | **Gana**: pagáis sólo lo que usáis. | Pagáis la GPU aunque nadie pregunte. |
| Volumen alto y sostenido | El coste crece sin techo. | **Gana** a partir del umbral. |
| Datos que no pueden salir | Requiere acuerdo y revisión legal. | **Gana**: el dato no sale de casa. |
| Máxima calidad | **Gana**: tier alto disponible ya. | Los libres van por detrás. |
| Equipo pequeño | **Gana**: cero operación. | Necesitáis MLOps. |
| Modelo ajustado a lo vuestro | Limitado a lo que ofrezca el proveedor. | **Gana**: control total de los pesos. |

**Idea clave.** Lo habitual en una startup: API para arrancar y medir; servidor cuando el volumen y el equipo lo justifiquen.

# Lo que os lleváis de este tema.




**Concepto 1.** Un agente es un modelo dentro de un bucle con herramientas y permisos. El harness es el programa que lo ejecuta (Claude Code, Cowork, opencode) y determina la calidad más que el modelo.

**Concepto 2.** Workflow cuando conocéis los pasos; agente cuando no. Subir de arquitectura cuesta dinero y predictibilidad.

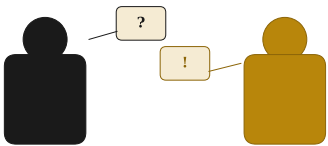
**Concepto 3.** El contexto es un presupuesto compartido, y el historial de la sesión es lo único que crece solo.

**Concepto 4.** Tier, caché y batch son tres palancas independientes que juntas recortan la factura un 84 % sin tocar la funcionalidad.

**Concepto 5.** El servidor propio se decide por volumen, residencia del dato o pesos propios; se calcula con el punto de equilibrio.

**Para la práctica.** Montar un agente sobre un repositorio real y medir, con `/context` y `/cost` primero y con la calculadora

# Pausa para debatir: ¿dejaríais que un agente mandase un correo a un cliente?




**La pregunta.** Vuestro agente redacta respuestas a reclamaciones. Funciona bien el 97 % de las veces. ¿Lo dejáis enviar sin revisión humana?

**Posición A.** Sí, con un modelo de riesgo: enviar solo por debajo de cierto importe y auditar una muestra.

**Posición B.** No. Un correo a un cliente es irreversible, y el 3 % restante es el que acaba en prensa.

**Reglas.** Cinco minutos, dos turnos por bando, móviles boca abajo. Yo modero y no participo.

# Hasta el Tema B.

*RAG, MCP, multi-agente y adaptación.*

**Eduardo C. Garrido-Merchán**

*ecgarrido@comillas.edu*

